FINAL COURSE PROJECT - REINFORCEMENT LEARNING

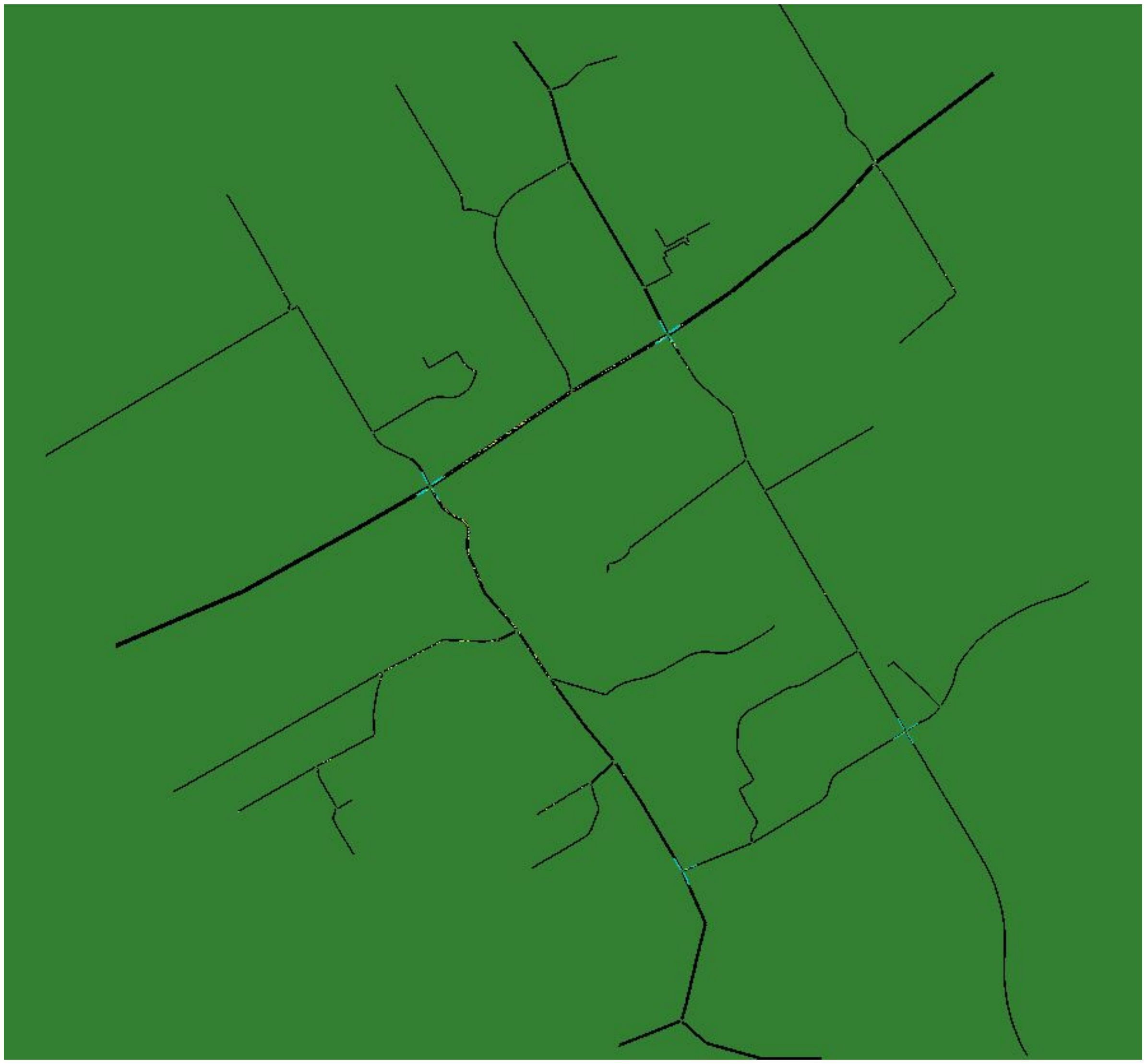
Adaptive traffic control system using n-step SARSA

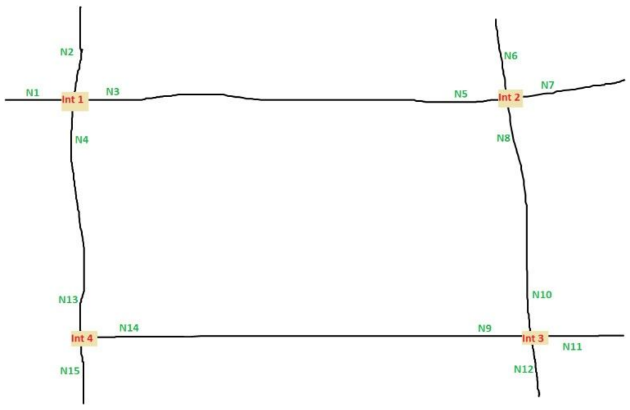
Akshay Kekuda
R. Anirudh
Mithun Krishnan

Problem Statement:

Traffic management is an area of great interest where lots of research takes place. We aim to apply Reinforcement Learning to solve the traffic management problem. Our aim is to find an optimal policy to route traffic in order to reduce congestion and waiting times. For our project we have taken a real-world 4-junction road network from Texas, USA using OpenStreetmap.

We use Simulation of Urban Mobility (SUMO) software to simulate traffic scenarios. Using SUMO, traffic patterns are generated and are used to test the Reinforcement Learning algorithm.



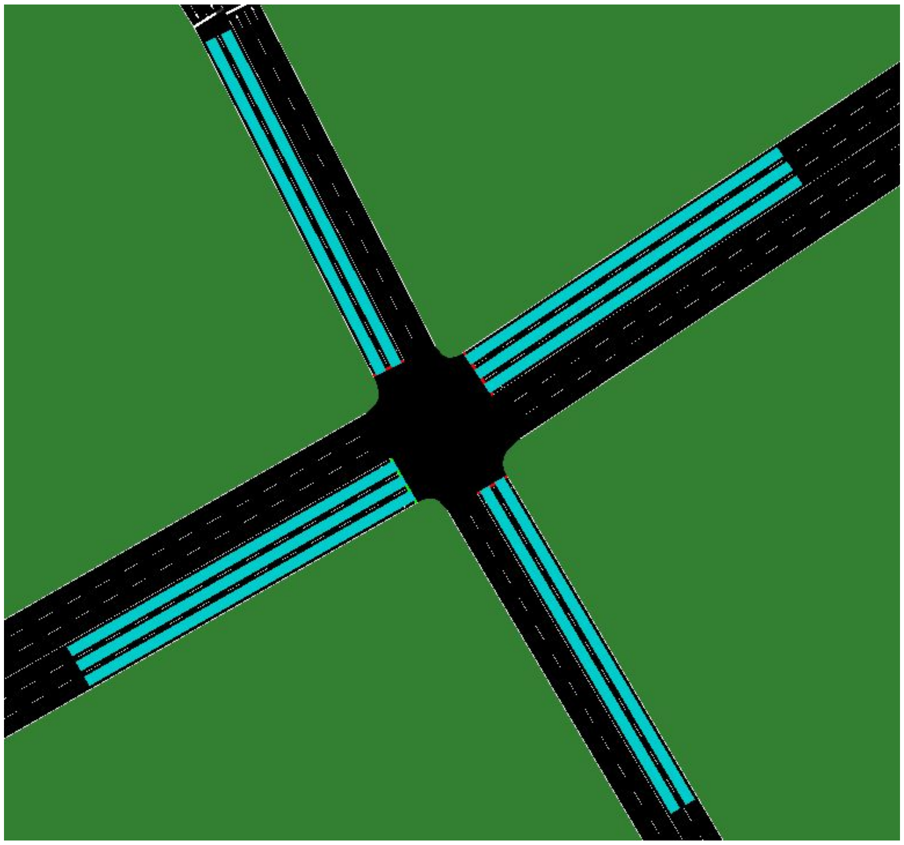


The above figure shows the traffic signals that are operating in our setting. There are a total of 15 signals across the 4 intersections. In our work, intersections are operated cyclically (INT1 -> INT2 -> INT3 -> INT4 -> INT1) at a gap of 16secs. At each intersection, the decision on which signal to operate is made by the RL agent. The goal of the RL agent is to minimize Queue Lengths, Waiting Time, Time Loss and to disperse traffic through the network in the least time. We have compared our algorithm with the following existing algorithms and have obtained better results:

1. **Static Signalling:** Typical system present in cities. Signals are operated in a fixed manner (every 16s, signals change) in a round robin fashion for each intersection. In this setting, the order was (n1-n5-n9-n13)-(n2-n6-n10-n14)-(n3-n7-n11-n15)-(n4-n8-n12-n13)--------- and so on
2. **Longest Queue First:** At every intersection, a decision is made based on the side which has the maximum queue length and green is given for that side. Again, as per our setting, we operate on intersections cyclically.

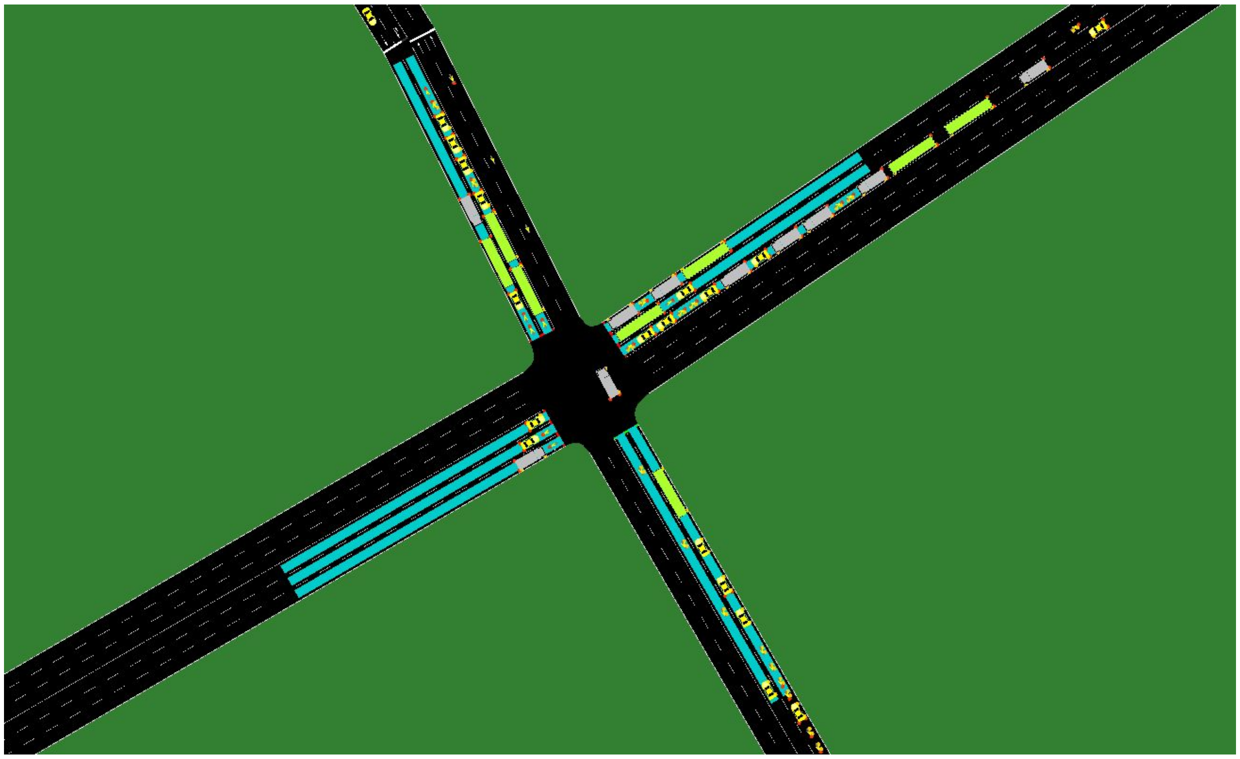
SUMO environment details:

1. **Use of induction loops:** To detect the amount of traffic waiting at the signals, we use Lane area detectors (E2 detectors) covering upto 70mts from the intersections. These detectors output the percentage of its area on which vehicles are standing.



2. **Description of lanes:** Our network consists of roads of one to three lanes. All the individual lanes are 10ft wide. The main arterial roads are 1.5km to 2km in length.

3. **Types of vehicles:** The traffic consists of 4 types of vehicles. The traffic distribution is modelled close to urban traffic. Bikes (41%), Cars (37.5%), Trucks (12.5%), Buses (8.3%). These vehicles have different speeds, acceleration and braking time parameters.



4. **Traffic density and pattern:** The number of vehicles entering the network follows a Binomial distribution with n=5 & p=0.15, which closely resembles a real world scenario. The vehicles are also distributed according to the number of lanes in a road, with higher traffic in multi-lane roads.

Algorithm:

(i) On-policy control - Differential semi-gradient n-step SARSA

The algorithm described below has been used for this work. Ours is a **continuing task**. We try to estimate the optimum policy for our network.

To generalize n-step bootstrapping, n-step version of TD error is used. We started by generalizing the n-step return:

$$G_{t:t+n} \doteq R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{n-1}R_{t+n}+\gamma^{n}Q_{t+n-1}(S_{t+n},A_{t+n}),\ \ n \geq 1, 0 \leq t < T-n$$

to its differential form, with function approximation:

$$G_{t:t+n} \doteq R_{t+1}-\bar{R}_{t+n-1}+\cdots+R_{t+n}-\bar{R}_{t+n-1}+\hat{q}(S_{t+n},A_{t+n},\mathbf{w}_{t+n-1})$$

where $\bar{R}$ is an estimate of r(π), n≥ 1, and t +n < T. If t + n ≥ T, and $G_{t:t+n} = G_t$.

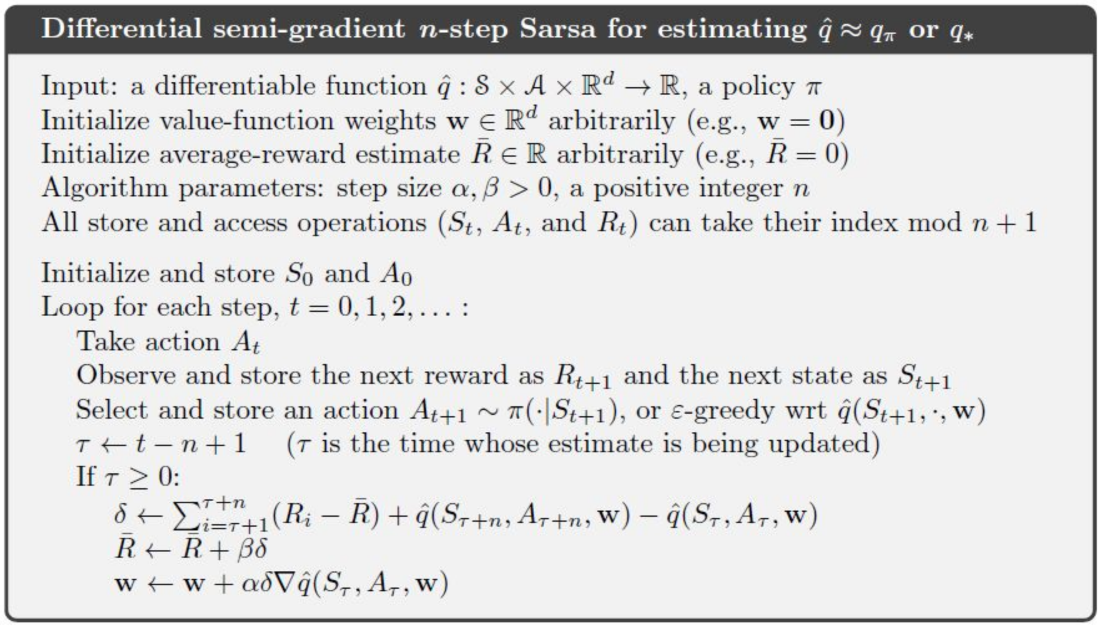
The n-step TD error is then

$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w})$$

Now we applied the semi-gradient Sarsa update:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

Pseudocode for the complete algorithm is given below:



Differential semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_\pi$ or q_*

```
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, a policy $\pi$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average-reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)
Algorithm parameters: step size $\alpha, \beta > 0$, a positive integer $n$
All store and access operations ($S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Initialize and store $S_0$ and $A_0$
Loop for each step, $t = 0, 1, 2, \ldots$ :
    Take action $A_t$
    Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$, or $\varepsilon$-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
    If $\tau \geq 0$:
        $\delta \leftarrow \sum_{i=\tau+1}^{\tau+n}(R_i - \bar{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$
        $\bar{R} \leftarrow \bar{R} + \beta\delta$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S_\tau, A_\tau, \mathbf{w})$
```

In a continuing task, TD error can increase w to infinity unless its expected value is zero. By subtracting the TD average estimation, the expected value of our update value is zero and w cannot go to infinity. TD error is a biased estimation of the average reward (with a bias that goes to zero as the number of updates goes to infinity assuming that every state can be reached from every state since the average reward is independent of the state action combination with which we start).

(ii) Q value estimation using function approximation:

We use linear function approximation to estimate Q values. What we want in practice are Q(s,a) values, due to the key fact that

$$\pi(s) = \arg_a \max Q^*(s,a)$$

which avoids a costly sum over the states. This will cost extra in the sense that a function with respect to a in addition to s should be used, but again, this is not usually a problem since the set of actions is typically much smaller than the set of states.

To use Q-values with function approximation, we need to find features that are functions of states *and* actions. This means in the linear function regime, we have

$$Q(s,a) = \theta_0 \cdot 1 + \theta_1 \phi_1(s,a) + \cdots + \theta_n \phi_n(s,a) = \theta^T \phi(s,a)$$

However, it's usually a lot easier to reason about features that are only functions of the states. So the "dimension scaling trick" which makes the distinction between different actions explicit is used. To make this clear, imagine an MDP with two features and four actions. The features for state-action pair (s,a$_i$) can be encoded as:

$$\phi(s,a_1) = \begin{bmatrix} \psi_1(s,a_1) \\ \psi_2(s,a_1) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_2) = \begin{bmatrix} 0 \\ 0 \\ \psi_1(s,a_2) \\ \psi_2(s,a_2) \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_3) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \psi_1(s,a_3) \\ \psi_2(s,a_3) \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_4) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \psi_1(s,a_4) \\ \psi_2(s,a_4) \\ 1 \end{bmatrix}$$

Details of the Algorithm used:

States: State of the environment corresponds to the congestion levels in the 15 sides. We have experimented with the following state vectors:

1. Queue lengths corresponding to 15 sides (length of state vector = 15)

2. Percentage occupancy corresponding to 15 sides (length of state vector = 15) **[Used finally, as it was found to best describe the system]**
3. Percentage occupancy corresponding to 15 sides and dependent percentage occupancy among 15 sides. Suppose x and y are the percentage occupancies of two sides in the network, the dependency state feature is calculated as x*y.

Actions: We have a total of 15 actions, which correspond to the signal that is to be made Green.

Reward: We have experimented with two different reward structures:

1. Queue lengths based reward. Reward at time t is defined as:

   $$R_t = -\sum_{j=1}^{M} q_{t,j}.$$

   Where $q_{t,j}$ is the queue length of side j at time t **[Used finally, as it gives consistent results]**
2. Reward based on the change in percentage occupancy of a side after an action is taken for that side. If traffic from side N_i goes to side N_j, and if both sides are crowded, then N_i shouldn't be opened. We penalize such actions.

This kind of reward structure makes our traffic system a centralised one, thus making it better than LQF and static signalling systems.

ϵ-greedy Action Selection: During n-step SARSA training, next action A_{t+1} is chosen ϵ-greedily. We found that the algo demanded initial exploration more than exploitation. So for the first run, we maintain ϵ=1 to encourage full exploration. We have experimented with the following ϵ decay strategies:

1. Constant ϵ=0.1
2. ϵ decay following 1/n, where n is the run index of simulation
3. ϵ decay following exp(-n), where n is the run index of simulation **[Used finally, as it gives consistent results]**

Feature vector:
DImension of feature vector = Length of state vector x Size of action space + 1 (bias term). Here in addition to the states and actions, we introduce a bias term in the feature vector to account for the state info not captured by any of the features in the feature vector. In our case, the feature vector length is 15 x 15 + 1 = 226.

n-step SARSA parameters:
These parameters were finalized after experimentation with several values. These values provided consistent good results.
Alpha = 0.1 / (run + 1), where run is the run index of simulation

Beta = c * alpha
n = 3
c = 2
Epsilon decay for e-greedy action selection: Exponential decay of epsilon for every run according to : exp(-run)

Training: 20 runs, where each run is around 16,000 simulation time steps. Each simulation time step equals 1sec. So each run is around 4.44hrs. And total training is equivalent to 3.7 days.

We train the RL agent under different traffic patterns each run and obtain the 226-length trained weights for Linear function approximation. Based on these trained weights, we run a SARSA Live algorithm to evaluate the performance, that chooses greedy actions based on the trained weight for a simulation.

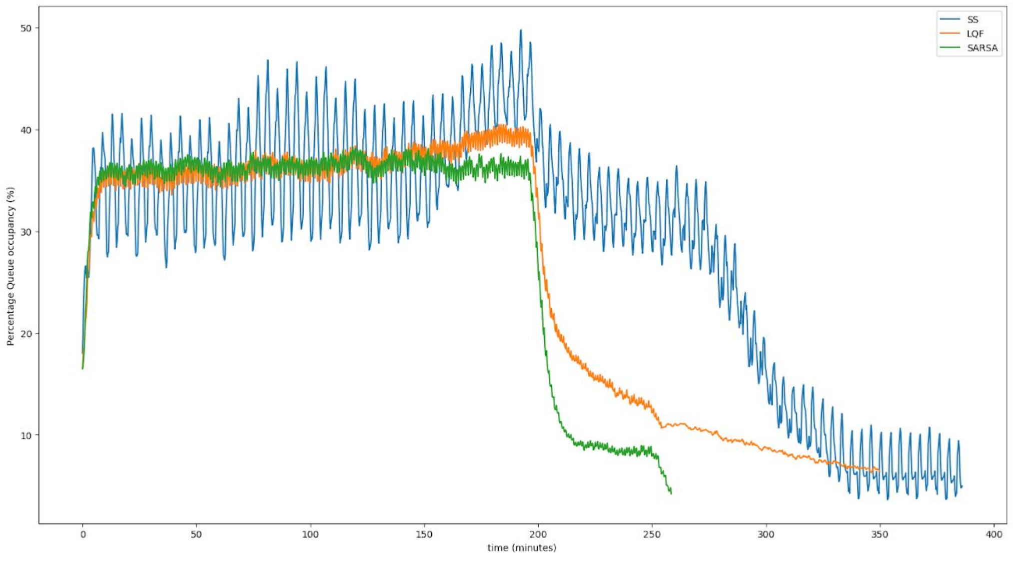
Traffic Patterns: We have tested the RL agent with randomly generated traffic patterns and observed that its performing better than LQF and Static signalling systems. We test in a heavy traffic scenario (p = 0.75).
(SUMO Traffic parameters: p = 0.75, e=12000, binomial=5, -L)

Results:

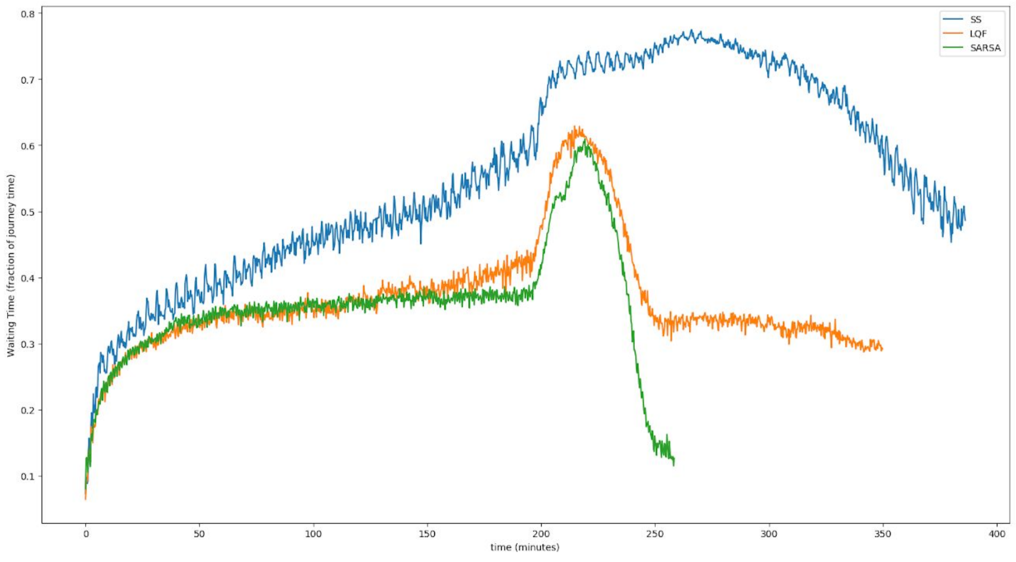
The algorithms are compared using the following metrics:

1) **Percentage queue occupancy:** Each signal has a lane area detector, which outputs the percentage of its area occupied by standing vehicles. This metric is the average occupancy of the 15 signals.
2) **Waiting Time:** This is the fraction of the journey time that a vehicle spends waiting at signals.
3) **Time Loss:** The time lost by a vehicle due to it driving below the ideal speed (due to slowdowns at signals etc.). This is taken as a fraction of the journey time.
4) **Dispersion time:** Time needed to disperse the traffic present in the road network. To measure this metric, traffic is allowed to flow for about 3.3hrs, after which, the time needed to empty the network is observed.

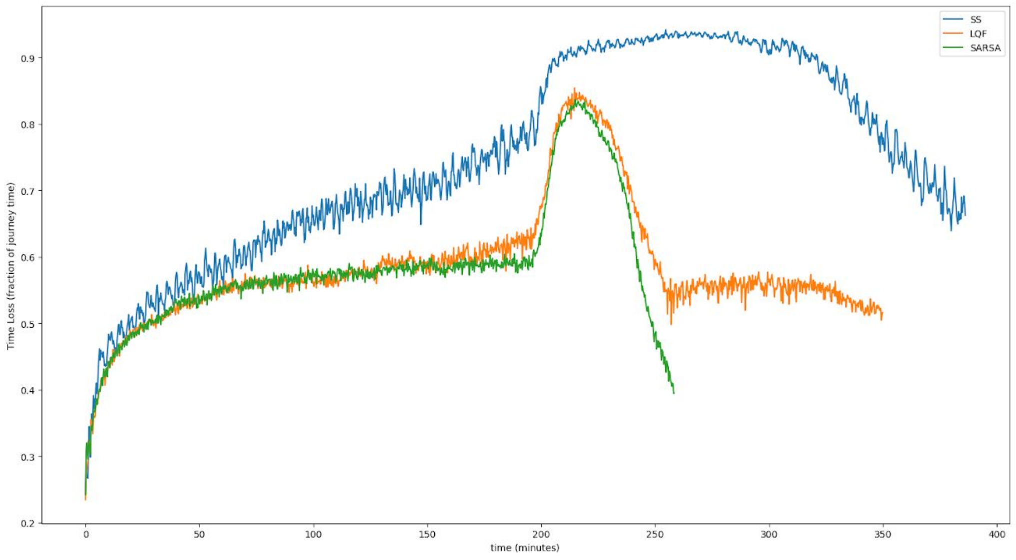
Queue Length



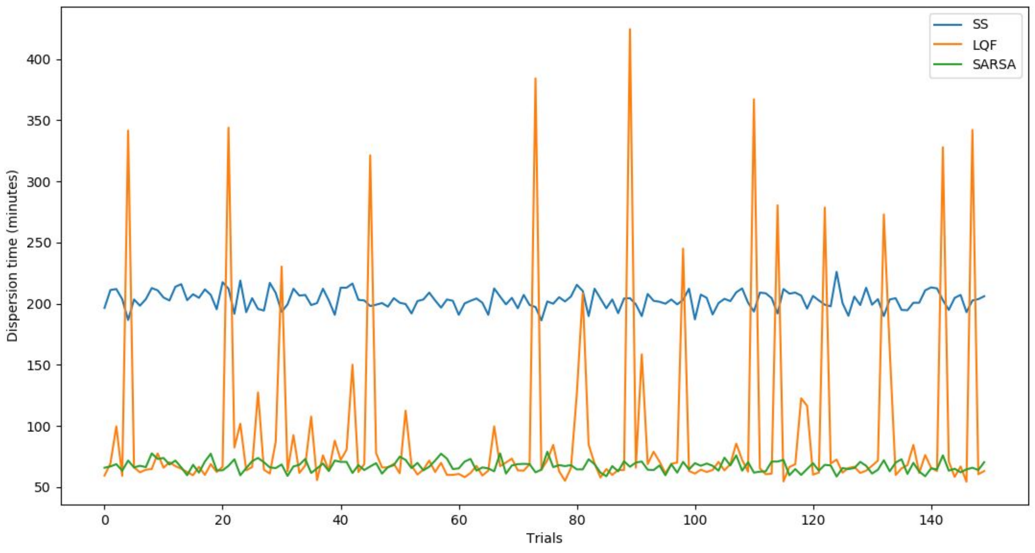
Waiting Time



Time Loss



Time to disperse traffic



Conclusion:

Below table lists the performance of the n-step SARSA algorithm for various n and c values. To evaluate this algorithm we have set a DeadLock Threshold of 1050 time steps. This threshold is a parameter under Dispersion Time metric. Any live trial that crosses this threshold is flagged as deadlock. We have computed the Deadlock Percentage i.e the number of times SARSA algorithm entered deadlock state, by running the SARSA LIVE algorithm for 150 trials. The n step SARSA algorithm achieves 0% deadlock with n =3 and c=2. Also as n increases beyond 3, the algorithm fails to converge to the optimal policy, thereby increasing the deadlock percentages. The algorithm achieves good performance for 1<=n<5. In

c \ n	**1**	**2**	**3**	**4**
0.1	**88**	**65**	**73**	**66**
0.5	**57**	**49**	**45**	**34**
1	**0.8**	**0.13**	**0.067**	**14**
1.5	**0.45**	**0.15**	**0.01**	**17**
2	**0.6**	**0.08**	**0**	**13.6**
2.5	**0.6**	**0.06**	**0.02**	**5.5**
3	**1.07**	**0.08**	**3.6**	**24.6**
3.5	**1.4**	**0.09**	**12.4**	**36.7**
4	**2.9**	**1.4**	**9**	**28.8**
4.5	**9**	**5.5**	**5.7**	**15.5**
5	**9**	**6.04**	**5.9**	**26**

From the graphs shown under the Results section, it can be seen clearly that n-step SARSA outperforms Static Signalling [SS] and Longest Queue First [LQF] in all the metrics.

Static signalling, the typical traffic signalling followed in cities shows very poor performance. It has a high variance since it is not adaptive.

With respect to **Queue length**, the network takes about 150mins to reach steady state traffic flow. Once steady state is achieved, n-step SARSA starts outperforming both SS and LQF. It shows a **5.5%** improvement as compared to LQF and **52.94%** improvement as compared to SS in steady state. During the dispersion phase (200mins onwards), n-step SARSA is able to disperse the traffic **31.5%** faster than LQF and **275%** faster than SS.

With respect to **Waiting Time**, the network takes about 100mins to reach steady state traffic flow. Once steady state is achieved, n-step starts outperforming both LQF and SS. It shows a **14.2%** improvement as compared to LQF & **133%** improvement as compared to SS in the steady state and dispersion phases. This shows that the vehicles reach their destinations faster in case of n-step SARSA.

With respect to **Time Loss**, the network takes about 100mins to reach steady state traffic flow. Once steady state is achieved, n-step SARSA starts outperforming both SS and LQF. It shows a **10.2%** improvement as compared to LQF & **81.8%** improvement as compared to SS in the steady state and dispersion phases.

The **Dispersion Rate** of traffic during dispersion phase is very high for n-step SARSA, as can be seen in the steep descents of the n-step graphs for all the metrics.

Finally, with respect to **Dispersion time**, we observe that LQF has multiple spikes in its dispersion times and SS has a dispersion time of 200 mins. These are called deadlock situations, wherein, the traffic comes to a complete halt due to congestion. As can be seen from the plots, n-step SARSA never enters (**0%**) a deadlock state due to its ability to predict and avoid such situations. Thus n-step SARSA has the lowest dispersion times on all trials.

References:

1. Reinforcement Learning: An introduction by Richard S. Sutton and Andrew G. Barto, Chapter 10.
2. Reinforcement Learning With Function Approximation for Traffic Signal Control: http://www.cse.iitm.ac.in/~prashla/papers/2011RLforTrafficSignalControl_ITS.pdf
3. Diagnosing Reinforcement Learning for Traffic Signal Control: https://arxiv.org/pdf/1905.04716.pdf
4. Adaptive Traffic SIgnal Control: Exploring Reward Definition for Reinforcement Learning https://www.sciencedirect.com/science/article/pii/S1877050917309912
5. SUMO User Documentation: https://sumo.dlr.de/docs/SUMO_User_Documentation.html